DSP Tutorial III
Avoiding Resource Overutilization in FPGAs

Larry Doolittle

Lawrence Berkeley National Laboratory

LLRF13, Lake Tahoe, October 2013

Index

Resources

Finite Resources in an FPGA

- Logic cells (small look-up tables paired with flip-flops)
- Multipliers (18x18 bit signed)
- Memory
- Clocks
- Routing
- Timing

This is a hardware talk!

- No code examples
- Nominally portable between Brand A and Brand X.

Up-front knowledge

Know your hardware!

- I can show you how to efficiently map DSP ideas to hardware.

Know your tools!

- How to express that hardware to your favorite tool chain. (not covered in this talk)

Know your tools and the hardware!

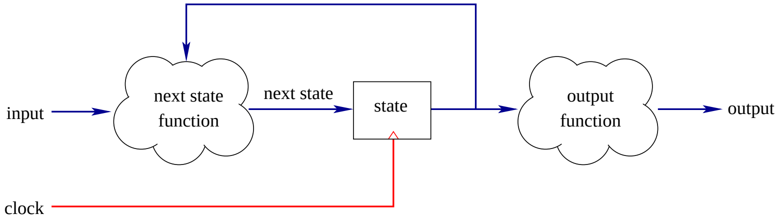
- Write HDL, visualize gates

Vocabulary item: 'make timing' or 'timing closure'

If you have a "huge" FPGA, feel free to ignore this talk ... until the project scope creeps enough that your FPGA no longer feels so huge.

Logic Cell

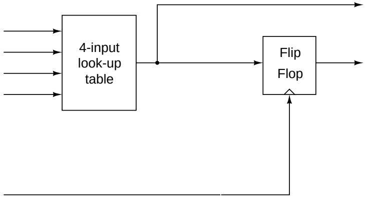
Finite State Machine:



(Moore formulation, synchronous clocking)

Logic Cell

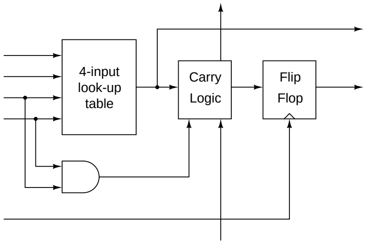
Simplest possible concept:



And, or, xor, multiplex, two-way mux with force-to-constant

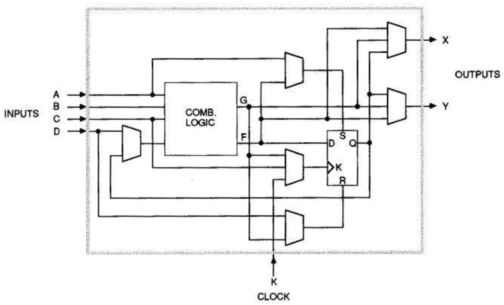
Logic Cell

Add a carry chain:



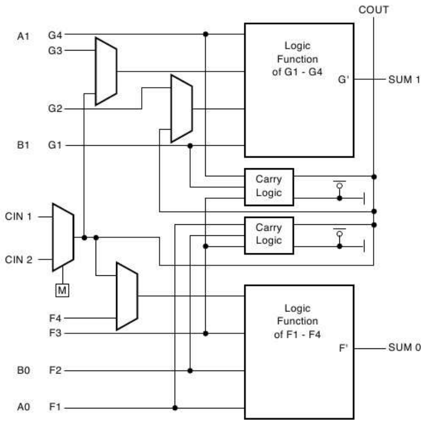
Add, subtract, gated-add, add/sub

Logic Cell

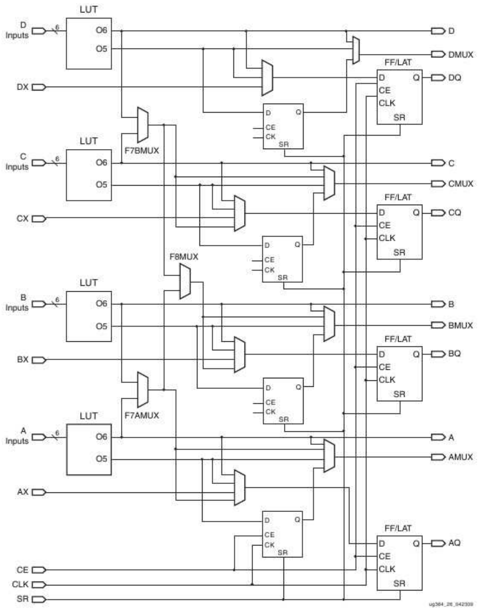


First commercial FPGA (XC2064) in 1985 had 64 of these cells

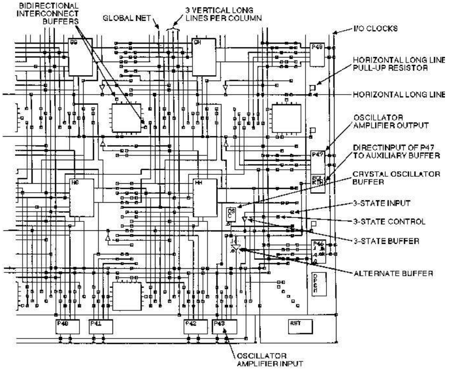
Logic Cell



Modern carry chain (XC4000) from 1991


LUT
D Inputs
6
O6
O5
DX
F7BMUX
C Inputs
CX
F8MUX
B Inputs
BX
F7AMUX
A Inputs
AX
CE
CLK
SR
D
Q
CE
CK
SR
FF/LAT
D
DMUX
DQ
C
CMUX
CQ
B
BMUX
BQ
A
AMUX
AQ
ug364_28_042309

Routing


BIDIRECTIONAL INTERCONNECT BUFFERS
GLOBAL NET
3 VERTICAL LONG LINES PER COLUMN
I/O CLOCKS
HORIZONTAL LONG LINE PULL-UP RESISTOR
HORIZONTAL LONG LINE
OSCILLATOR AMPLIFIER OUTPUT
DIRECT INPUT OF P47 TO AUXILIARY BUFFER
CRYSTAL OSCILLATOR BUFFER
3-STATE INPUT
3-STATE CONTROL
3-STATE BUFFER
ALTERNATE BUFFER
OSCILLATOR AMPLIFIER INPUT

Well documented routing from 1998 (XC3000).
Routing needs scale super-linearly with chip size; routing hardware is now much more capable, complex, and less well documented.

Clocks

Handful of well-defined domains: good
Clock-enable on registers: good
Low latency data path involving multiple clock domains: bad
Gated clocks: terrible

Clock domain crossing

- events
- Gray codes
- data - low throughput (flagged register)
- data - high throughput (DPRAM, flagged blocks or FIFO)

As FPGAs get larger and faster, demands on the clock tree get harsher. Newer chips deprecate global clocks; clock-domain crossing is needed between zones. Think of it as several chips in one package (sometimes literally true). Tools and design practices must be adjusted to cope.

Clocks

Generally our designs have at least two clock domains:

- High throughput DSP has no wiggle-room on its speed, has to match ADC
- Communications (e.g., Ethernet, USB, VME, PCI)

Many unavoidable reasons to increase from the minimum, e.g., White Rabbit leaf node needs one or two more domains to function correctly.

If you have lower throughput DSP stuff to do, *and* wish to re-use someone else's "IP" (e.g., a soft core) that may not make timing with your main DSP clock, you may need to generate a subharmonic clock for that subcircuit. That will count as a new clock domain! Example:

- Z80 from OpenCores at 40 MHz
- LM32 at 80 MHz

Clocks

Only a finite number of clock domains are supported on any given FPGA!

Miscellaneous “low speed” logic, e.g., a serial DAC or thermometer, doesn’t need a separate clock domain! If the logic is clean, it will make timing at the primary clock rate. Simply put clock-enables on all its registers to slow down its sequencing.

- Saves the clock domains for essential uses
- Simplifies attachment to the rest of the chip’s logic.

Memory Hierarchy

On-chip memory

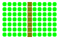
- Small, granular, flexible, fast:
  - Xilinx: 16 to 64 bits each, one-bit access
  - Altera: 32x20 or 64x10 in some chip families
- Block memory: dual-port, 4K to 36K bits each, 1 to 36 bit data ports
- Large block: single-port (rare)

On-chip memory: just inferred from HDL

Off-chip memory: slow, more difficult to use and simulate

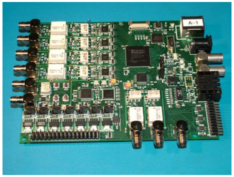
Up-front knowledge

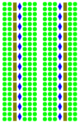
year	*system*	*FPGA*	×	*logic*	*RAM*	
2002	SNS-Interim	XC2S200	0	4704	7	←
2006	LLRF4	XC3S1000	24	15360	54	
2011	Fermi@ELETTRA	XC5VSX95T	640	58880	1098	
2012	Struck SIS8300	XC5VLX50	48	28800	216	
2013	LLRF4.6	XC6SLX45	58	54576	260	
2013	Red Pitaya	XCZ7010	80	35200	60	



Up-front knowledge

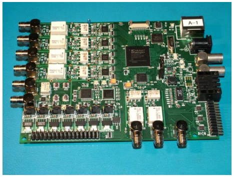
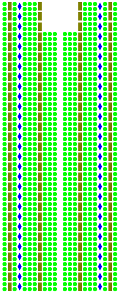
year	*system*	*FPGA*	×	*logic*	*RAM*	
2002	SNS-Interim	XC2S200	0	4704	7	
2006	LLRF4	XC3S1000	24	15360	54	←
2011	Fermi@ELETTRA	XC5VSX95T	640	58880	1098	
2012	Struck SIS8300	XC5VLX50	48	28800	216	
2013	LLRF4.6	XC6SLX45	58	54576	260	
2013	Red Pitaya	XCZ7010	80	35200	60	





Up-front knowledge

year	*system*	*FPGA*	$\times$	*logic*	*RAM*	
2002	SNS-Interim	XC2S200	0	4704	7	
2006	LLRF4	XC3S1000	24	15360	54	
2011	Fermi@ELETTRA	XC5VSX95T	640	58880	1098	
2012	Struck SIS8300	XC5VLX50	48	28800	216	
2013	LLRF4.6	XC6SLX45	58	54576	260	$\leftarrow$
2013	Red Pitaya	XCZ7010	80	35200	60	

Logic Reduction

Logic

Engineering philosophy:

- A register that doesn't get a "random" new bit every clock-cycle is under-utilized

Push the speed-area tradeoff to small and slow:

- Use bit-serial techniques

Multiplex data going through high speed parallel data paths:

- Soft cores can be considered a special, limiting case of multiplexed data paths, adding programmability.
- Smaller, simpler, faster approaches are often better. Look for DSP with SIMD (single instruction multiple data) character.

ASICs are sometimes power-dissipation-limited instead of gate-limited.
Leads to what is known as "dark silicon."

Multipliers

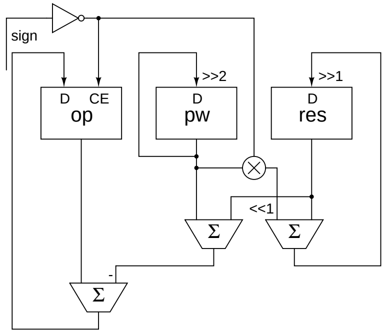
Dedicated multipliers on FPGAs are a great resource: fast, effective, well supported by synthesizers.

If you run out and you have multiplications that don't need full bandwidth, shift-and-add works pretty well! Small, clocks fast, can be encapsulated into a module that's not hard to use.

An $N \times N$ bit multiplier takes $N + 1$ cycles with $3N$ logic cells.

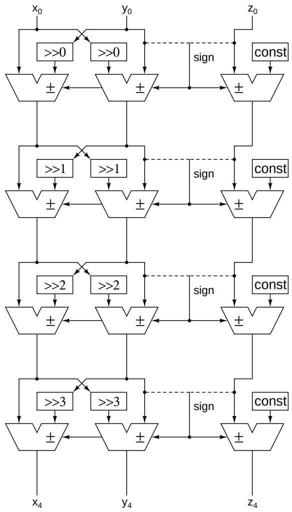
Square Root

Integer square root in $\sim$ 4.5 LUTs per bit, one cycle per two bits,
e.g., 158 LUTs and 17 cycles for 32 bits
Fits the shift-and-add paradigm



(initialization muxes not shown)

CORDIC

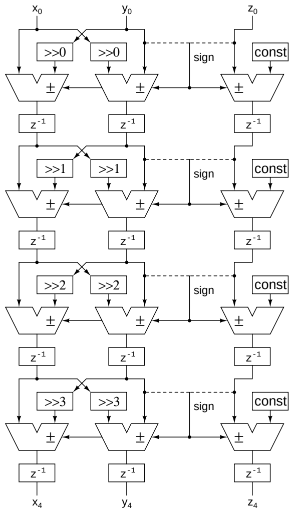


Jack Volder - 1959
Elaborate series of shift-and-add/sub
Each stage applies the matrix

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & \pm 2^{-n} \\ \mp 2^{-n} & 1 \end{pmatrix} \cdot \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$

Four unrolled stages shown.

CORDIC



Jack Volder - 1959
Elaborate series of shift-and-add/sub
Each stage applies the matrix

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & \pm 2^{-n} \\ \mp 2^{-n} & 1 \end{pmatrix} \cdot \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$

Four unrolled pipelined stages shown.

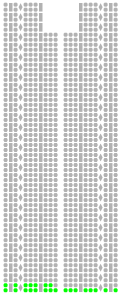
CORDIC

Unrolled-parallel CORDIC area scales as n^2 with the number of bits accuracy.

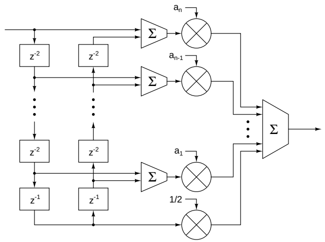
method	*LUTs*	*cycles*	*performance metric*
serial, bit-serial	71	512	28
serial, 2-bit-serial	95	256	41
unrolled, parallel	1481	21	32
unrolled, parallel, pipelined	1481	1	675

CORDIC

Full-featured, pipelined unrolled, CORDIC with 114 dB SNR

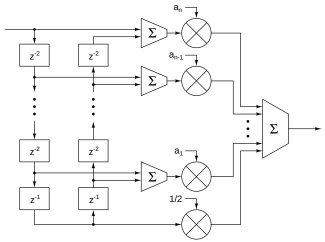


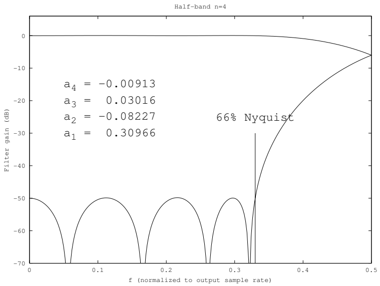
Reduced-multiplier filters



$\times 2$ for I and Q...

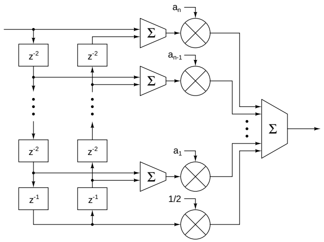
Reduced-multiplier filters

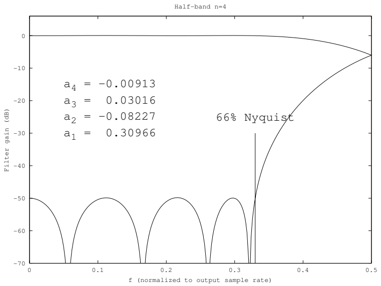


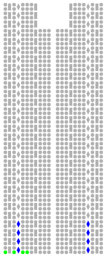


$\times 2$ for I and Q...

Reduced-multiplier filters

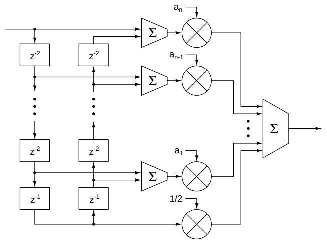


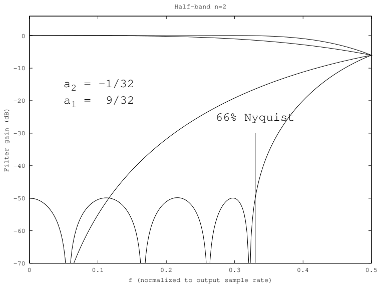




$\times 2$ for I and Q...

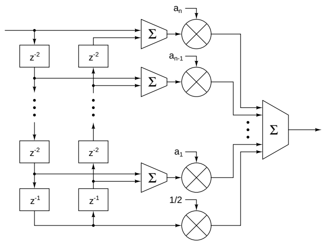
Reduced-multiplier filters

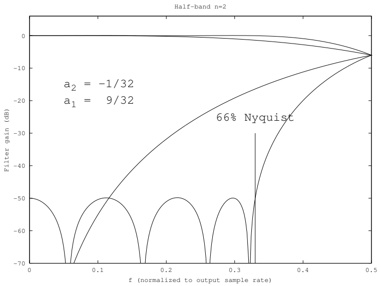


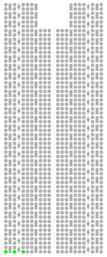


$\times 2$ for I and Q...

Reduced-multiplier filters





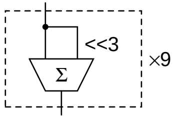


×2 for I and Q...

Reduced-multiplier filters

In general, constant-multiplier-based filters can be converted to a series of adders. Sometimes good results can be achieved with only a few bits of multiplier.

Synthesizers can do the substitution for you (and may resort to this if the FPGA runs out of “real” multipliers), but they have to do it literally, without re-optimizing the filter for this implementation, so their results are needlessly large and slow.



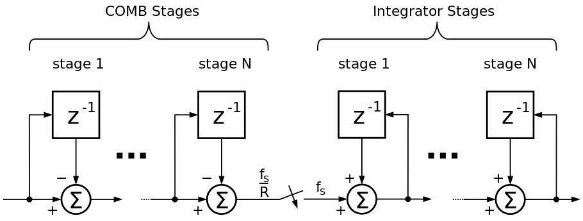
CIC filters

CIC (Cascaded-Integrator-Comb) N-th order filter of length RM:

$$\left(\sum_{k=0}^{RM-1} z^{-k}\right)^N = \frac{(1-z^{-RM})^N}{(1-z^{-1})^N}$$

“This equation shows that even though a CIC has integrators in it, which by themselves have an infinite impulse response, a CIC filter is equivalent to N FIR filters, each having a rectangular impulse response.”

- Matthew P. Donadio



CIC filters

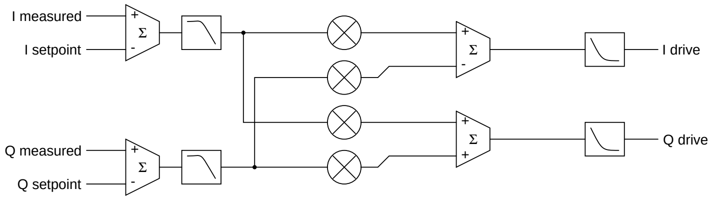
"Since all of the coefficients of these FIR filters are unity, and therefore symmetric, a CIC filter also has a linear phase response and constant group delay."

- Matthew P. Donadio

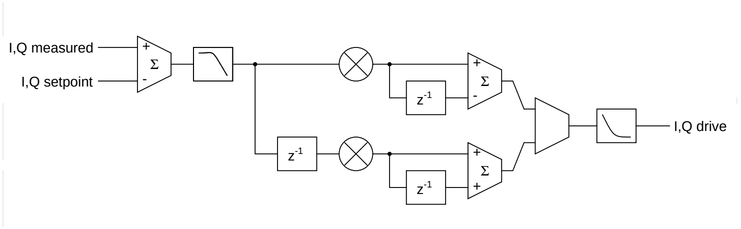
If the rate-change step is made programmable, the filter passband behavior changes too. This property makes CIC filters a great match to signal processing at the entrance to waveform memory, sometimes followed by one or more half-band filters.

Designers of CIC filters have to pay attention to system gain (RM^N) and bit growth.

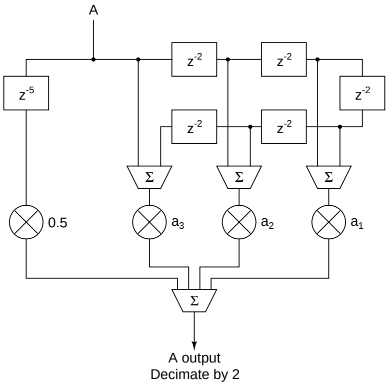
IQ path

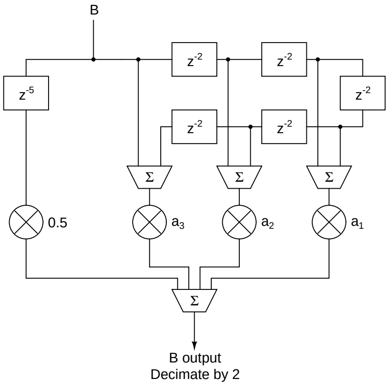


IQ path

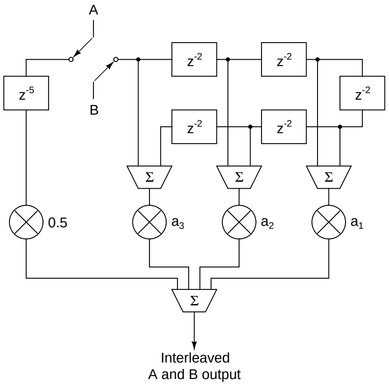


Half-band Decimator



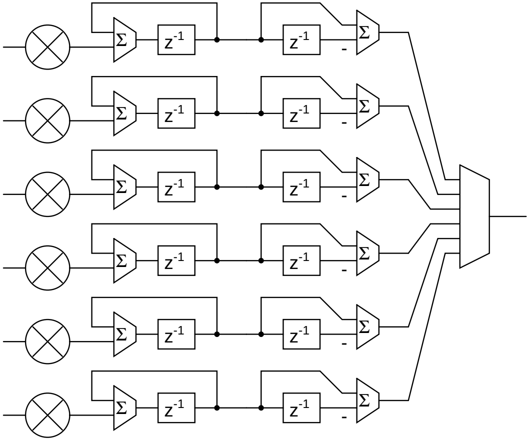


Half-band Decimator



CIC Decimator

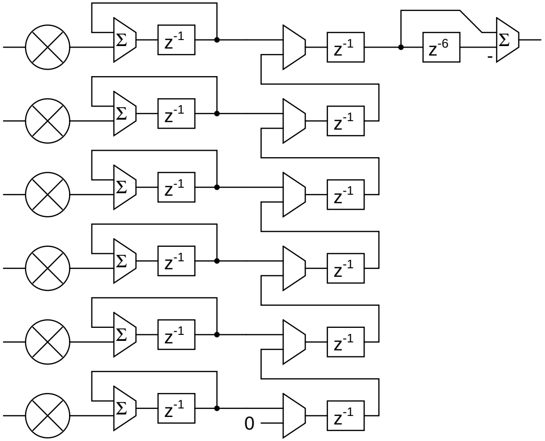
Initial concept:



∼4 cells per bit per channel, multiplexer a challenge to route for speed

CIC Decimator

Implementation with conveyor belt:



$\sim$2 cells per bit per channel, routing and timing easy and scalable

Soft Cores from OpenCores

Sturgeon's law: "Ninety percent of everything is crap"

16 Bit Microcontroller	Wbc	?
16-bit CPU based on Caxton Foster's Blue		LGPL
16-bit Open uRISC core Processor		LGPL
1664 microprocessor		Others
4004 CPU and MCS-4 family chips		Others
6502VHDL		?
68hc05		?
68hc08		?
8-bit microcontroller with extended peripheral set		LGPL
8051 core	Wbc	?
8080 Compatible CPU	Done	?
ae18	Done Wbc	LGPL
aeMB	Done Wbc	LGPL
ag_6502 soft core with phase-level accuracy	Done	GPL
AltOr32 - Alternative Lightweight OpenRisc CPU	Done	LGPL
Alwcpu - A light weight CPU	Done Wbc	LGPL

Soft Cores from OpenCores

Sturgeon's law: "Ninety percent of everything is crap"

Amber ARM-compatible core	Done Wbc	LGPL
ao68000 - Wishbone 68000 core	Wbc	BSD
Apollo Guidance Computer NOR eMulator		GPL
Aquarius	Wbc	GPL
ASPIDA sync/async DLX Core	Wbc	?
Atlas Processor Core	Wbc	GPL
AVR Core		?
AVR HP, Hyper Pipelined AVR Core	Done	LGPL
AVRtinyX61core	Done	LGPL
AX8 mcu		?
CF State Space Processor		?
ClaiRISC - runs 12bit opcode PIC family.	Done	?
Codezero OpenRISC Port		GPL
Confluence OpenRisc 1000		?
copyBlaze	Wbc	LGPL
CortexI		LGPL

Soft Cores from OpenCores

Sturgeon's law: "Ninety percent of everything is crap"

Cowgirl	Done	?
cpu6502_tc - R6502 Processor Soft Core	Done	GPL
cpu65c02_tc - R65C02 Processor Soft Core		GPL
Data Flow Processor		?
Diogenes: Student RISC System		?
ecpu		GPL
Educational 16-bit MIPS Processor	Done	LGPL
Educational RISC Processor		?
ELM Embedded Processor		Others
Encore		LGPL
FORTH processor with Java compiler	Done	LGPL
HC11 Compatible - Gator uProcessor	Done	LGPL
HiCoVec - a configurable SIMD CPU		GPL
HIVE - a 32 bit, 8 thread, 4 register/stack hybrid	Done	Others
HPC-16	Done	LGPL
HyperMTA		?

Soft Cores from OpenCores

Sturgeon's law: "Ninety percent of everything is crap"

Ion - MIPS(tm) compatible CPU		LGPL
JOP: a Java Optimized Processor	Wbc	GPL
K68		?
KLC32		LGPL
Lattice 6502		LGPL
LEM1_9	Done	LGPL
Leros: A Tiny Microcontroller for FPGAs	Done	BSD
Lightweight 8051 compatible CPU	Done	LGPL
Lightweight 8080 compatible core	Done	GPL
LocationPU		?
M1 Core	Wbc	GPL
MB-Lite	Wbc	LGPL
McAdam's RISC Computer Architecture		GPL
MCPU - A minimal CPU for a CPLD	Done	GPL
MicroRISC II		?
MicroSimplez	Done	LGPL

Soft Cores from OpenCores

Sturgeon's law: "Ninety percent of everything is crap"

Mini-Risc core		?
Minimal PDP8/L implementation		LGPL
miniMIPS		LGPL
Mips-FaultTolerant	Done	LGPL
MIPS32 Release 1	Done	LGPL
mips789		?
mipsr2000	Done	LGPL
MIPS_enhanced		LGPL
MPX 32-bit CPU		LGPL
myBlaze		LGPL
Natalius 8 bit RISC		LGPL
Navré AVR clone (8-bit RISC)	Done	GPL
nCore		GPL
Next 80186 processor		LGPL
NextZ80	Done	LGPL
nnARM core	Wbc	?

Soft Cores from OpenCores

Sturgeon's law: "Ninety percent of everything is crap"

oks8	Wbc	GPL
OoOPs - Out-of-Order MIPS (TM) Processor		LGPL
Open8 uRISC	Done	BSD
OpenCores54x DSP	Wbc	?
OpenCPU32		LGPL
OpenFire Processor Core		?
openMSP430	Done	BSD
OpenRISC 1000	Done Wbc	LGPL
OpenRisc 1200 HP, Hyper Pipelined OR1200 Core	Done	LGPL
OpenRISC 2000		LGPL
pAVR		?
PDP-11/70 CPU core and SoC	Done	GPL
PDP-8 Processor Core and System		GPL
Pepelatz MISC		LGPL
Plasma - most MIPS I(TM) opcodes	Done	Others
PPX16 mcu		?

Soft Cores from OpenCores

Sturgeon's law: "Ninety percent of everything is crap"

qrisc32 wishbone compatible risc core		LGPL
r2000 Soc	Wbc	LGPL
Raptor64		LGPL
Reduced AVR Core for CPLD		LGPL
RISC Microcontroller		?
risc16f84	Done	Others
RISC5x	Done	LGPL
RISC_Core_I		?
RISE Microprocessor		?
RTF65002	New Wbc	LGPL
rtf8088		LGPL
S1 Core	Done Wbc	GPL
SAYEH educational processor		LGPL
scARM - A SystemC ARM		?
Scarts Processor		LGPL
Small x86 subset core		LGPL

Soft Cores from OpenCores

Sturgeon's law: "Ninety percent of everything is crap"

Software Aided Wishbone Extension for PicoBlaze	Done Wbc	BSD
Storm Core (ARM7 compatible)	Wbc	GPL
SXP (Simple eXtensible Pipeline) Processor		?
system11		?
System68		?
T400 μController	Done	GPL
T48 μController	Done	GPL
T51 mcu	Wbc	LGPL
T65 CPU		?
T6507LP	Done	GPL
T80 cpu		?
TG68 - execute 68000 Code	Done	LGPL
The Neptune Core		?
Theia: ray graphic processing unit	Wbc	GPL
Tiny Instruction Set Computer		?
Tiny64		?

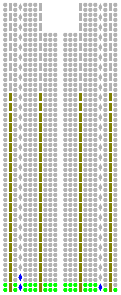
Soft Cores from OpenCores

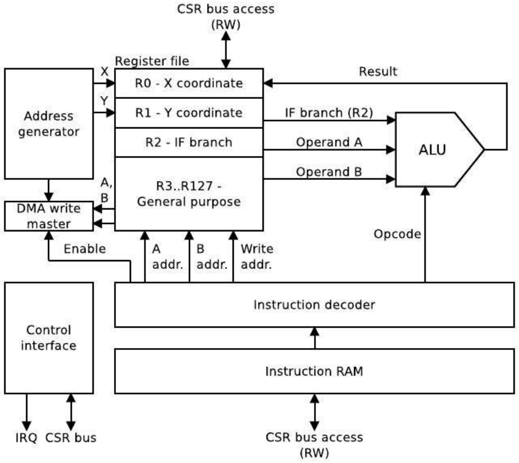
Sturgeon's law: "Ninety percent of everything is crap"

tiny8		?
TinyCPU		BSD
TotalCPU	Done	?
turbo 8051	Wbc	LGPL
TV80		BSD
UCore	Wbc	?
vhdl core of IC6821	Done	GPL
VTACH - Bell Labs CARDIAC reimagined in Verilog		LGPL
Wishbone BFM	Done Wbc	LGPL
Wishbone High Performance Z80	Done Wbc	?
Y80e - Z80/Z180/eZ80 compatible processor	Done	BSD
YACC-Yet Another CPU CPU		?
Yellow Star		?
z80control		GPL
Zet - The x86 (IA-32) open implementation	Wbc	GPL
ZPU - 32 bit CPU with GCC toolchain	Done Wbc	?

Soft Cores

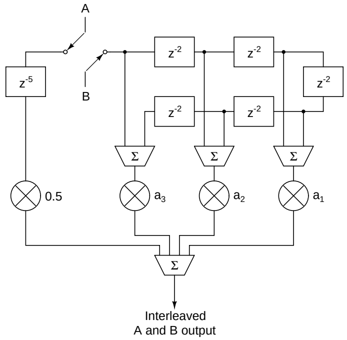
Full 32-bit gcc-targeted LM32 soft core as configured with 80K of RAM



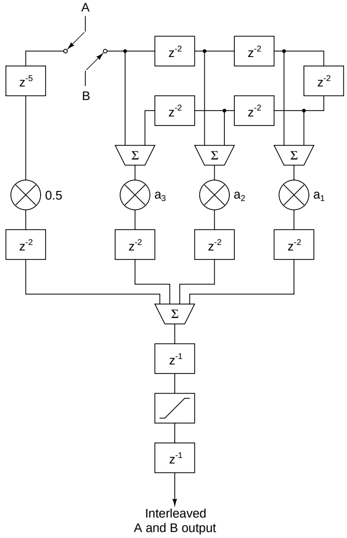


Milkymist Programmable Floating Point Unit - fast, small, dense code, effective for its target application, innovative, architecturally related to "Unified Shaders" in a modern GPU ... but like those GPUs, relatively tough to program.

Latency implications



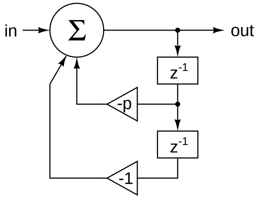
Latency implications



Reasonable speed multipliers take one or two clock cycles

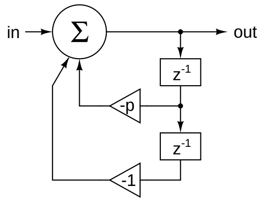
General trend (including ADCs, FPGA, communications) is for throughput to improve more than latency, as technology advances.

Latency implications



Beautiful ideal pole at $\cos^{-1}(-p/2)/T$ ...

Latency implications



Beautiful ideal pole at $\cos^{-1}(-p/2)/T$ ...
but totally unrealizable with real-world multiplier latency.

Conclusions

Please simulate your designs! Example report from CORDIC test:

```
Check of x,y,theta->x,y
test covers 15958 points, maximum amplitude is 90325 counts
peak error 1.25 bits, 0.0010 %
rms error 0.36 bits, 0.0003 %
PASS
vvp -n cordicg_tb +op=1 > cordic1.dat
Check of x,y,theta->r,theta
test covers 7979 points, maximum amplitude is 129001 counts
peak error 1.06 bits, 0.0008 %
rms error 0.36 bits, 0.0003 %
PASS
Check of downconversion bias
test covers 6102 points
averages 0.027 -0.007
PASS
```

Links

Xilinx FPGA history
`http://www-inst.eecs.berkeley.edu/∼cs294-59/fa10/resources/Xilinx-history/Xilinx-history.html`

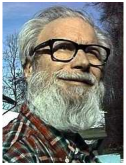
Altera FPGA history
`http://www-inst.eecs.berkeley.edu/` *sim*`cs294-59/fa10/resources/Altera-history/Altera-history.html`

Milkymist SoC
`http://www.milkymist.org/mmsoc.html`

Conclusions

"Engineering is the art of making what you want from things you can get."

– Jerry Avins



The way to avoid resource overutilization is to understand the hardware capability, and map your DSP needs effectively onto that hardware.

Save factors of two wherever you can.

Design quality counts: gives you margin in speed and area.

Thank You!